Sensitivity Analysis of a Parallel Particle Swarm Optimization Clustering Algorithm for Multi-objective Optimization

MOHAMMAD M. HAMDAN
Department of Computer Science
Faculty of Information Technology
Yarmouk University
Irbid 21163
JORDAN

Abstract: Parallel bio-inspired algorithms have been successful in solving multi-objective optimisation problems. In this work, we discuss a parallel particle swarm algorithm with added clustering for solving multi-objective optimisation problems. The aim of this work is to perform sensitivity analysis of the parallel particle swarm algorithm. We need to see how the added parallelism improves the overall execution time. Also, looked at the effect of different strategies for population initialisation (such as mutating current set of leaders, random population and lookup in archive for nearest points using geometric calculation). The results show that using different migration frequencies for scattering reduced the overall overlap between processors. Results regarding how clustering and gathering affect performance metrics are also reported.

Key-Words: Particle Swarm Algorithms, Parallel Algorithms, Migration, Clustering, Multi-objective Optimisation, Sensitivity Analysis,

Received: January 15, 2024. Revised: April 17, 2024. Accepted: June 19, 2024. Published: July 29, 2024.

1 Introduction

Real-world problems, have many objectives to be optimized at the same time. In most cases, it is not easy to solve the problem as the objectives are clashing with each other. The name Multi-objective optimization problems (MOPs) has been given to such cases, [1], and can be formally (for a minimization problem) defined as:

$$Min\, f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), ..., f_d(\overrightarrow{x})$$

where $\overrightarrow{x} \in \Omega \subseteq \Re^n$ is a vector of decision variables. It is difficult to find a unique solution vector $\overrightarrow{x}$ that minimizes all objectives at the same time. Therefore, the solution is a set called *non-dominated solutions*. A dominance relation denoted by "$\prec$" can be defined as follows, [1]:

$$(\overrightarrow{x_1} \prec \overrightarrow{x_2}) \Leftrightarrow \begin{array}{l} \forall i \in \{1, ..., d\}, f_i(\overrightarrow{x_1}) \leq f_i(\overrightarrow{x_2}) \\ and\, \exists i \in \{1, ..., d\}, f_i(\overrightarrow{x_1}) < f_i(\overrightarrow{x_2}) \end{array}$$

The Pareto dominance equation will define the minimal elements as Pareto-optimal. Also known as *Pareto-optimal set*. When plotted in the objective space they form the true Pareto-Front (PF_{true}). Solving MOPs is not easy but algorithms attempt to generate PF_{known} that is a good approximation of PF_{true}.

Biologically-inspired algorithms such as evolutionary algorithms have managed solving optimization problems. For example, swarm intelligence algorithms, [2], are very competetive optimizers for combinatorial and optimization problems. Two famous algorithms are Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), [3]. PSO is a population-based algorithm that has interacting boids or agents. It has been hybridized with other techniques as in [4].

There are several parallel algorithms for MOPs. Each has an approach to the parallelization process. Several techniques are found common among authors such as migration frequency, initial population, and interchange of individuals among processors. Therefore, it is important to study the various parameters that are used in these systems. In this study, we are proposing and discussing a parallel PSO for MOPs in addition to performing a thorough sensitivity analysis of its main parameters. We think this is something that is not commonly done in related parallel algorithms for MOPs.

In this work, we look at sensitivity analysis of a parallel PSO algorithm that uses clustering to scatter the Pareto Front among the parallel processors. In particular, we are interested in migration frequency as it relates to parallel behavior. The rest of the paper is organized as follows. In Section 2, background information is given about PSO. Section 3 presents a parallel PSO for MOPS. The results are shown and discussed in Section 4. Section 5 provides the summary and concludes the work.

2 Background

The following subsection outline, particle swarm algorithms for MOPs and presents an overview of parallel evolutionary algorithms for MOPS.

2.1 Particle Swarm Optimisation for MOPs

The Particle Swarm Optimisation (PSO) method is a well-known optimization algorithm in the area of biologically inspired computing. Here, the algorithm mimics flocks of birds when flying, [3]. Each individual (i.e particle) updates its location in the search by information transmitted from other particles.

It can be used to solve a general function *F(x)*, where *x* is a possible solution vector in a multidimensional space. Here, a group of individuals (potential solutions of *F(x)*, called boids) continously update their positions to reach required destination. The goal is to reach the global minima using data received. In this work, the classical PSO is integrated with a weight factor for restricting the velocity, [5]. The formulas shown below are used to update the the n*th* particle velocity and position:

$$V_{i+1}^n = wV_i^n + c_1r_1(P_L^n - x_i^n) + c_2r_2(P_g - x_i^n), \quad (1)$$

and

$$x_{i+1}^n = x_i^n + \alpha V_{i+1}^n. \quad (2)$$

Here, α is the time step; P_L^n is the local best vector for n*th* particle, and P_g is the global best vector for all particles; *w* is inertia weight factor; c_1 and c_2 are social factors; r_1 and r_2 are random numbers (between 0 and 1);

Single-objective PSOs have been successfully extended to multi-objective optimization problems by: Adding an external archive for collecting non-dominated points. Adding a turbulence operator such as the highly-disruptive polynomial mutation. Replacing the global best particles with a set of leaders from an external archive. Managing the external archive by using an update mechanism and limiting the size of the external archive using crowding or clustering.

2.2 The SMPSO Algorithm

SMSPO, [6], stands for Speed-constrained Multiobjective PSO. It is state-of-the-art in the area of evolutionary multiobjective optimization. Its main features are in the use of a constriction coefficient to control the particle's velocity. The use of velocity constriction. The use of polynomial mutation rather than uniform/non-uniform mutation. Different ranges for C1 and C2 [1.5,2.5] and inertia weight set to 0.1.

2.3 Parallelisation of Evolutionary Algorithms

A number of approaches, [7], [8], [9], [10], [11], [12], have been proposed and carried out in the past regarding how to parallelize evolutionary algorithms. The main approaches are Master-worker, Island, Diffusion and hybrid models. Master-worker (also called global parallelization) model simply decodes and evaluates the fitness function of each individual of the population on the slaves. In the Island model, the main population is divided into subpopulations (or demes) and placed on different nodes. Each node has its sub-population and runs an EA as usual. It has two variants: with or without migration (i.e. sending individuals to other nodes). A similar approach is called cross-pollination as in [13].

In the previous two approaches the parallelization was on the level of the population. However, in the diffusion model, [10], (also called pollination models, [14], fine-grain parallel evolutionary algorithms and neighborhood model) the parallelism is exploited on the individuals' level. Here, each individual is placed on a processor and forms a neighborhood structure with few individuals from its local environment.

It is possible to mix two approaches or more in a hybrid parallel evolutionary algorithm as in [15]. Interesting work in this area is the work of [16], (in a system called pMOHypEA), where it is possible to structure the population from coarse-grain island models to fine-grained diffusion models using the idea of hypergraphs.

Every two generations (as set in the experiments) the local subpopulations on the remote processors are gathered by the master processor, clustered into *k* subpopulations using the clustering centroids, and redistributed to the remote processors. The clustering is applied to the current Pareto-front. It has two modes: 1) cluster the search space, and 2) cluster the objective space. Also, zone constraints are added to every processor using the constrained dominance principle, [17], to limit the subpopulations to their specific partition. However, it is possible to mark individuals as invalid in case they are assigned to another centroid than the subpopulation they belong to. From the results obtained by experiments, it is clear that objective space clustering outperforms search space clustering. Additionally, when the zone constraints are deactivated the results improved. This means that each processor can explore and exploit the entire objective space and processors can overlap considerably. Also, in the experiments they use a total population size of 600 individuals which might be needed for the clustering approach to work. No computational speedup results were reported as we think the process gathering, clustering, and redistributing the results could hinder the computational performance significantly. More on parallelisation approached can be found in [18], [19], [20]

3 A Parallel PSO for Multicriteria Optimisation

A PSO algorithm for multicriteria optimization has been developed as part of a project funded by the Ministry of Higher Education in Jordan. The algorithm was achieved by adding the following the single objective optimization PSO algorithm, [21]:

- An external archive for collecting the non-dominated points,
- Adding polynomial mutation as a turbulence operator to the PSO model,
- Replacing the global best particles with a set of leaders from the external archive chosen randomly,
- Limiting the size of the external archive by using a crowing distance operator,
- Managing the archive by using an update mechanism.

Algorithm 1 Outline of the Parallel PSO clustering-based multi-objective algorithm.

```
Each Processor $P_i$ will execute:
1) If the main archive is empty then
Define a random population
Else
Take one cluster (i) from the master processor
2) Produce new population around cluster (i)
3) Use a PSO algorithm
4) Collect Sub-archives from slave processors
5) Update the main archive (located on the main processor)
6) Apply the k-means algorithm to cluster archive into k-clusters
7) Repeat steps 1-6 Until the Terminate condition is True
```

The sequential PSO for MOP was used as part of parallel systems that are described in Algorithm 1. Each part was assigned to a different processor. Each processor generates a population around the given cluster and applies the PSO algorithm for multi-objective optimization. Such a kind of division of the objective space is novel and can achieve good results in terms of the quality of obtained solutions as different processors focus on different parts of the objective space in parallel. The overall design of the proposed system is shown in Figure 1. Each processor is running SMPSO in a multi-start approach. The processors perform scatter and gather operations according to the frequency of gathering. Once the leaders are gathered on the processor, they are clustered into *P* clusters. The clusters are then scattered to processors. Each processor will take its part as a leader. There is a need for a population for the leaders. Later in the experiments, we discuss how we can generate a population for the leaders using more than one technique. To visualize the run time behavior of the proposed system we can see Figure 2 as it illustrates how the main archive is clustered into 8 different pieces for DTLZ1 and DTLZ2.

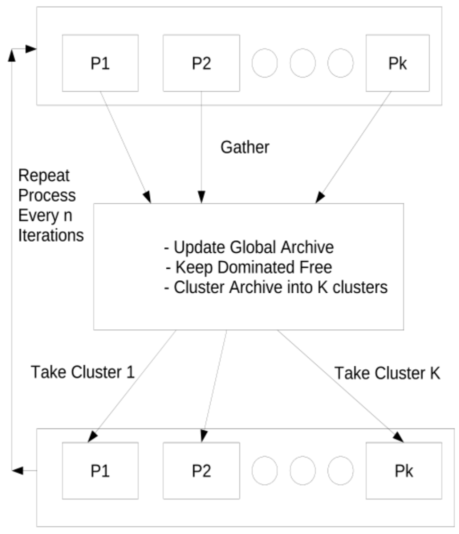


Fig. 1: Overall design of the proposed parallel PSO System for MOPs

4 Experiments and Results

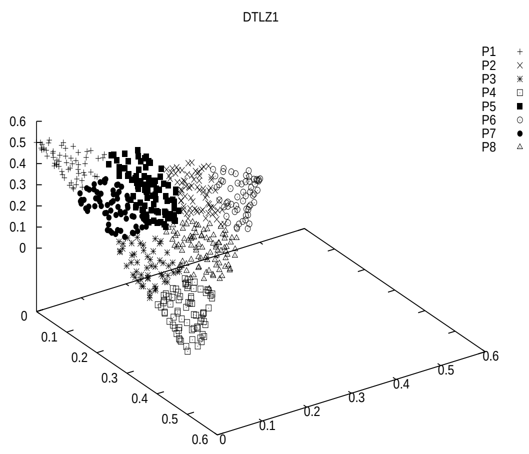
Here, thorough results are shown regarding the proposed parallel system.

4.1 Experimental Environment

To evaluate the system we used the following problems: ZDT1 (2D, convex Pareto front), ZDT3 (2D, disconnected Pareto front), DTLZ1 (3D) and DTLZ2 (3D). We used 25000 function evaluations and the results were averaged over 10 runs. To measure the quality of the final obtained solutions we used Inverted Generational Distance (IGD), Hypervolume (HV) and Spread. As a turbulence operator, we used the Polynomial Mutation. The entire system was implemented using C + MPI. Experiments were conducted on a cluster of PCs.

4.2 Adding Parallelism

In the initial design of the system, only the root processor was responsible for generating the initial set of leaders. The other processors remain idle. Therefore, we parallelized this step using a static master-slave model. Here, all processors work in parallel to produce the initial set of leaders. Each processor will run



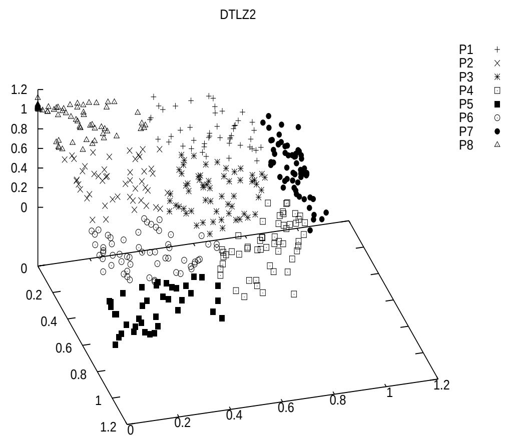


Fig. 2: The distribution of the Pareto front across 8 processors for DTLZ1 and DTLZ2

a sequential version of the base algorithm (SMPSO), then after a few iterations collect the results on one processor. The gain here is number of iterations will be divided equally among processors. The execution time improved as can be seen in Figure 3. Here, the execution time drops as we increase the number of processors.

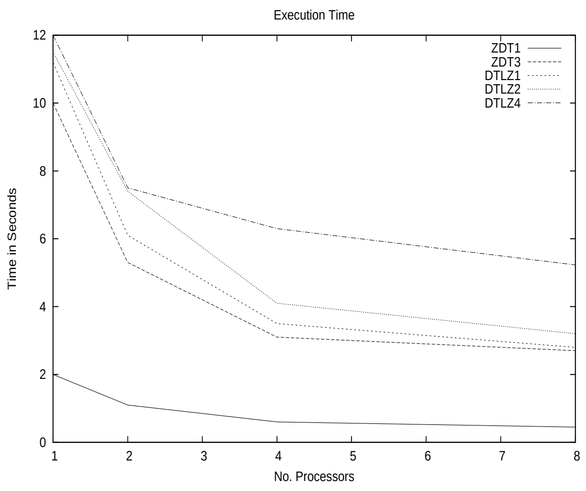


Fig. 3: Sequential vs Parallel Execution Time

.

4.3 Different Population Initialization Strategies

In this part, we need to figure out the best method for generating a new population for each processor after we perform gathering and clustering. Each processor will get its leaders to be used by the SMPSO algorithm. Different methods were proposed for this:

- perform polynomial mutation on the current set of leaders (called init 1 in the figures). This approach would work if there were many leaders to mutate.

- generate the population randomly as we do in the initial stage of the system (called init 2 in the Figures).

- The algorithm maintains an archive of current population (called init 3). When leaders are given, lookup/search in the archive for the nearest points (calculated geometrically) to the current leaders. The set of points will form the new population. This approach aims to generate the best set of individuals that shall follow the leader. If far away then lots of iterations will be needed to converge to leader. However, if very close to the leader then not lots of variation can happen (premature convergence). In an ideal world, having a mixture of points might work well.

Therefore, we performed an extensive set of experiments to evaluate the suggested three mechanisms for generating the population. Figure 4 and Figure 5 show HV and IGD results, respectively. It seems that certain problems prefer a given initialization strategy while other problems prefer other strategies. Therefore, to be on the safe side then generating the population randomly would be fine for most of the problems.

4.4 Migration Frequency

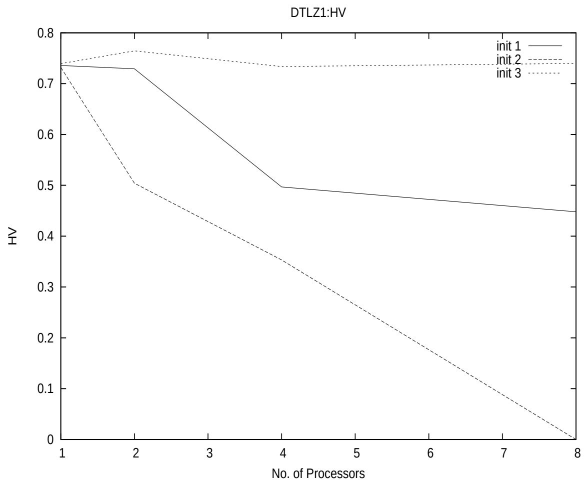
In this set of experiments, we would like to see the effect of the frequency of gathering, clustering, and scattering back to processors on the overall performance of quality metrics. Figure 6 shows the HV results and Figure 7 shows the IGD results.

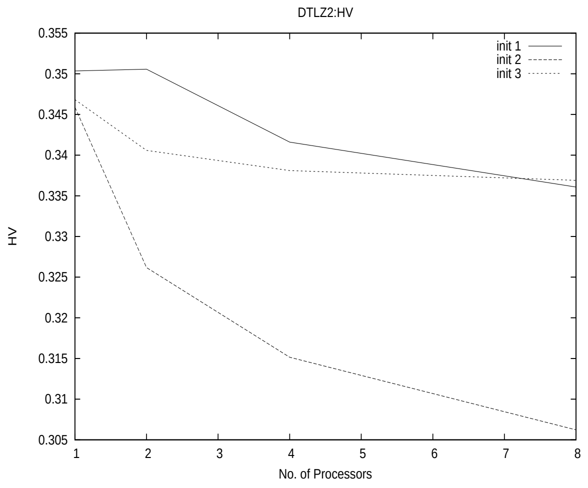
The main conclusion to draw from these results is that most of the problems prefer a high frequency of gathering. The gathering every 40 or 50 iterations did not perform as well as frequencies of 5, 10, and 20. When migration is performed frequently then clustering is applied to the collected population we can reduce the overlap among processors. Although this might add extra cost to communication overhead these are negligible compared to the extra gain in quality metrics. The issue now is that as migration is very often then it is expected to improve HV and IGD metrics. However, there is the cost for transmitting the population and it is expected that overall speedup will drop as more processors are used. Here, the users of the system need to understand the tradeoffs among speed and MOP metrics.

However, as more processors are used (beyond 4), the performance does not improve. On the contrary, it starts to drop. This is due to the large overlap among processors and smaller cluster sizes given to processors. Also as the total number of iterations is divided among processors, the base algorithm on each processor does not get enough function evaluations to progress well.

5 Summary and Further Work

The results obtained from the experiments are interesting. We can conclude the following: Adding parallelism to the first stage of the existing system does improve execution time. Different techniques for generating the neighborhood population are good. The suggested three approaches to answer this question. However, the random generation of the population did not do well. This is due to locality issues as the population is far away from the leader. It is advisable to either mutate the leader or maintain an archive of the old population then for a given leader search





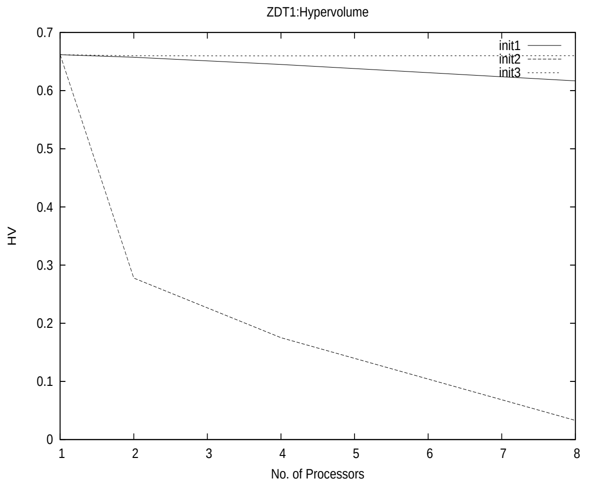


Fig. 4: Initialisation strategies effect on hypervolume metric

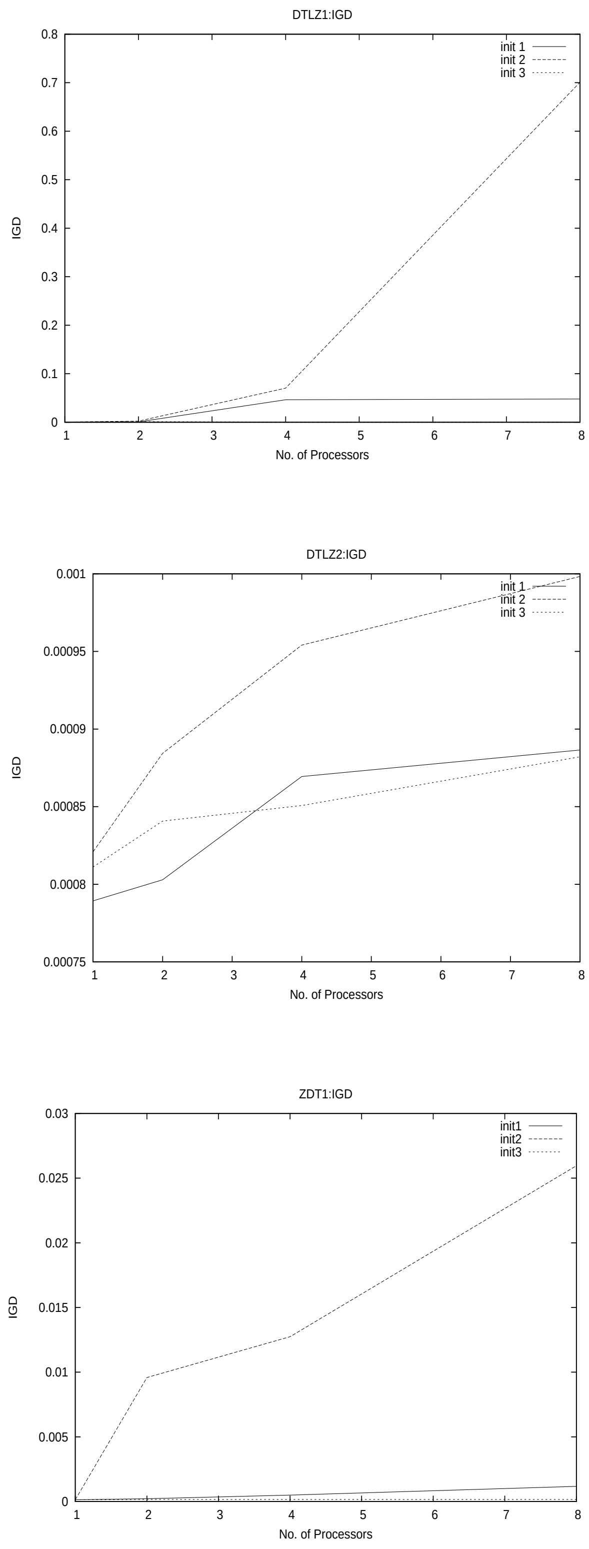


Fig. 5: Initialisation strategies effect on IGD metric

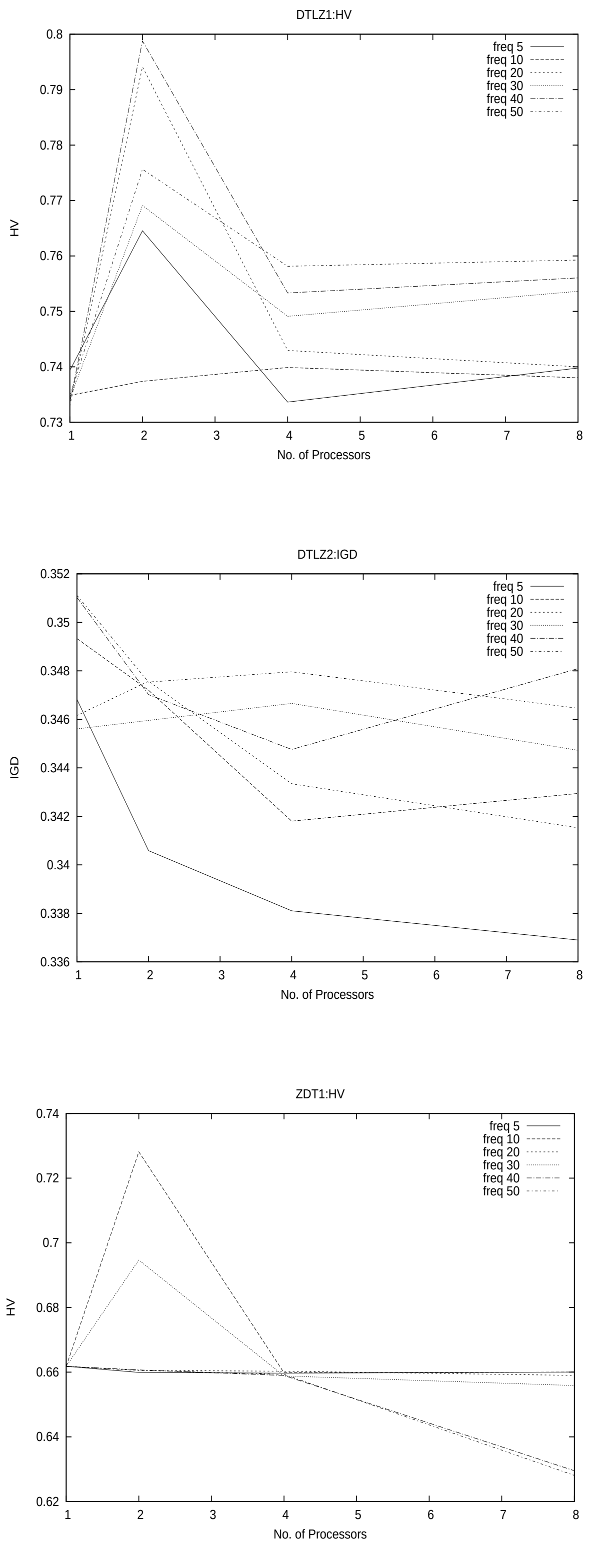
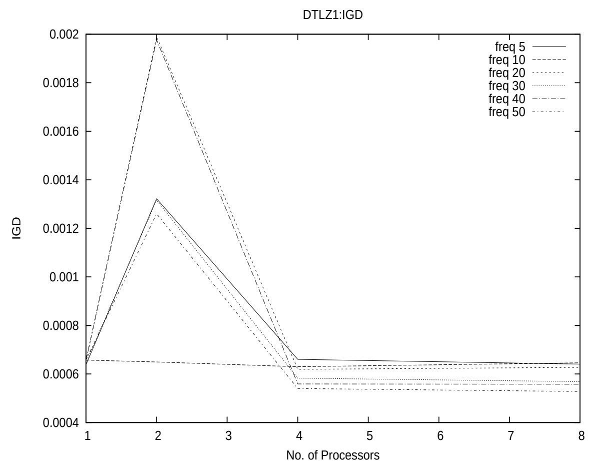
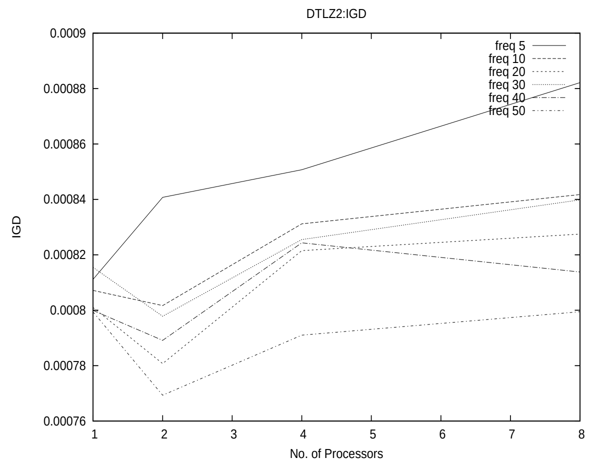


Fig. 6: Migration Frequency effect on hypervolume metric





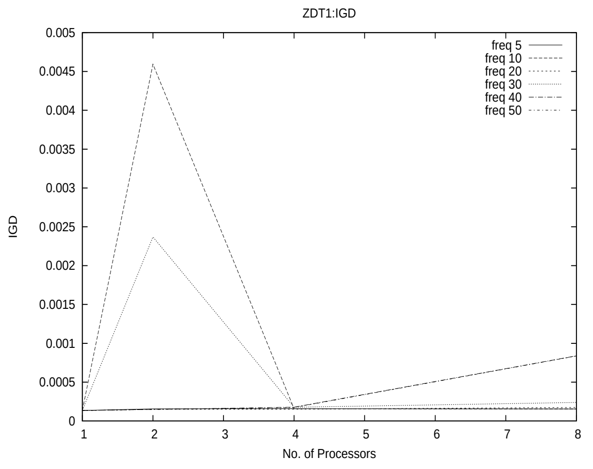


Fig. 7: Migration Frequency effect on IGD metric

for the closest points within the archive. The frequency of gathering and clustering the non-dominated points on other processors is important. This should be done quite often (preferably every 5, 10, or 15 iterations). Waiting for longer periods will cause lots of overlap of regions among processors and reduce the overall quality of metrics. The above issues could be problem-dependent. In other words, different problems may prefer different initialization and/or gathering frequency. For further work, the sensitivity analysis could be done for further studies.

References:

[1] Carlos A. Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[2] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[3] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. of the IEEE Int. Conf. on Neural Networks*, pages 1942–1948, Piscataway, NJ, USA, 1995.

[4] Abdallah Qteish, Mohammad Hamdan. Hybrid particle swarm and conjugate gradient optimization algorithm. *International Conference in Swarm Intelligence* pages 582–588. Springer, 2010.

[5] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 67–73, 1998.

[6] A.J. Nebro, J.J. Durillo, J. Garcia-Nieto, C.A. Coello Coello, F. Luna, and E. Alba. Smpso: A new pso-based metaheuristic for multi-objective optimization. In *Computational intelligence in miulti-criteria decisionmaking, 2009. mcdm '09. ieee symposium on*, pages 66 –73, 30 2009-april 2 2009.

[7] E. Alba and F. Chicano. On the behavior of parallel genetic algorithms for optimal placement of antennae in telecommunications. *International Journal of Foundations of Computer Science*, 16(2):343–359, 2005.

[8] E. Alba and J. M. Troya. A survey of parallel distributed genetic algo rithms. *Complexity*, 4(4):31–52, 1999.

[9] Enrique Alba and Jose M. Troya. Analyzing synchronous and asynchronous parallel distributed

genetic algorithms. *Future Generation Computer Systems*, 17:451–465, 2001.

[10] Sven E. Eklund. A massively parallel architecture for distributed genetic algorithms. *Parallel Computing*, 30:647–676, 2004.

[11] Wilson Rivera. Scalable parallel genetic algorithms. *Artificial Intelligence Review*, 16:153–168, 2001.

[12] Mohammad Hamdan, Gunter Rudolph, Nicola Hochstrate. A Parallel Evolutionary System for Multi-objective Optimisation. *IEEE Congress on Evolutionary Computation (CEC) 2020* pages 1–9. IEEE, 2020.

[13] Lucas A. Wilson and Michelle D. Moore. Cross-pollinating parallel genetic algorithm for multi-objective search and optimization. *Inter national Journal of Foundations of Computer Science*, 16(2):261–280, 2005.

[14] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Ma chine Learning*. Addison-Wesley, Reading, MA, USA, 1989.

[15] S. N. Sivanandam, S. Sumathi, and T. Hamsapriya. A hybrid parallel genetic algorithm approach for graph coloring. *International Journal of Knoweldge-Based and Intelligent Engineering Systems*, 9(3):249–259, 2005.

[16] Jorn Mehnen, Thomas Michelitsch, Karlheinz Schmitt, and Torsten Kohlen. pMOHypEA: Parallel evolutionary multiobjective optimization using hypergraphs. Technical Report Reihe CI-189/04, University of Dortmund, 2004. ISSN 1433-3325.

[17] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

[18] J. Branke, H. Schmeck, K. Deb, and M. Reddy. Parallelizing multiobjective evolutionary algorithms: Cone separation. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1952–1957, 2004.

[19] L. Bui, H. Abbass, and D. Essam. Local models - am approach to distributed multi-objective optimization. *To appear in Computational Optimization and Applications*, 2007.

[20] Antonio L. Jaimes and Carlos A. Coello Coello. MRMOGA: a new parallel multi-objective evolutionary algorithm based on the use of multiple resolutions. *Concurrency and Computation: Practice and Experience*, 19(4):397–441, 2007.

[21] A.J. Nebro, J.J. Durillo, J. García-Nieto, C.A. Coello Coello, F. Luna, and E. Alba. Smpso: A new pso-based metaheuristic for multi-objective optimization. In *2009 IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (MCDM 2009)*, pages 66–73. IEEE Press, 2009.

Acknowledgment: The author would like thank Jordan University of Science and Technology for hosting the sabbatical year.
The main author did the current work from design, experiments and writing.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The author contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

The work of this study was supported by a sabbatical leave from Yarmouk University.

Conflicts of Interest Nothing to declare.